A Verilog RTL Synthesis Tool for Heterogeneous FPGAs

Peter Jamieson and Jonathan Rose
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
Homogeneous FPGAs

Consist of:
1. Programmable logic and routing
   • soft logic fabric
   • Basic logic unit
   • Programmable Routing
Heterogeneous FPGAs

Consist of:

1. Programmable logic and routing
   - soft logic fabric
   - hard structures

2. Dedicated hard structures
   - e.g. multiplier
   - e.g. memory block
Fundamental Trade-off

• Hard structures provide benefit when used
  – Faster
  – Smaller
  – Consume less power
However...

- If hard structure *not* used
  - *wasted Silicon*
- Routing resources wasted
- 70-90% of FPGA area occupied by routing
Motivation of This Work

• Long term: derive more benefit from hard structures
  – By exploring the fundamental trade-off
• Academic CAD flows currently can’t target these structures
State-of-the-Art CAD Flows

• Existing Front-end Synthesis tools do target heterogeneous FPGAs
  – Altera’s Quartus
  – Xilinx ‘ISE
  – Mentor’s LeanardoSpectrum
  – Synplicity’s Synplify
  – Synopsys’ Design Compiler FPGA
  – Magma’s Blast FPGA
Goals of this Work

1. Front-end tool to map to hard structures
2. Achieve comparable results to Industrial Front-end Synthesis
3. Deliver open source for academic community
Our Tool

Called "Odin"

– maps Verilog HDL designs to heterogeneous FPGA architectures
– can interface with existing CAD flows:
  • Quartus
  • Modelsim
  • VPR (no heterogeneity)
– Can be used as front end to the following heterogenous CAD Flow
Heterogeneous FPGA CAD Flow

- Input:
  - HDL design

```verilog
module small (a, b, c, out);
  input[5:0] a, b, c;
  output [5:0] out;
  assign out = ({2'b00, a[2:0]} * b) + (b & ~c));
endmodule
```
Heterogeneous FPGA CAD Flow

- Parse HDL
  - Icarus creates an intermediate representation
Heterogeneous FPGA CAD Flow

- Elaboration [Odin]
  - Convert into a netlist
  - Preserve high-level Information

```
2'b00
a
*b
b
&
c
!+ out
```
Heterogeneous FPGA CAD Flow

• Optimize RTL [Odin]
  – Arithmetic Operations
  – Finite State Machines
  – Multiplexers

```
2'b00
```

```
a

b

&

*

+

out

c

!```
Heterogeneous FPGA CAD Flow

- Partial Mapping [Odin]
  - Identify high-level functions that map into hard structures
Heterogeneous FPGA CAD Flow

- Partial Mapping [Odin]
  - Bind to hard structures or soft fabric
Heterogeneous FPGA CAD Flow

• Output
  – Netlist targeting remainder of heterogeneous FPGA CAD flow

```verilog
module small (a, b, c, out);
  input[5:0] a, b, c;
  output [5:0]out;

  not(c, e2);
  and(b, e2, e3);
  lpm_mult(a[3:0],b,e1);
  lpm_add(e1,e3,out);
endmodule
```
Heterogeneous FPGA CAD Flow

- Input – HDL
- Odin
- Logic Optimize
- Technology Map
- Place
- Route
- Output – FPGA bitstream
Heterogeneous FPGA CAD Flow

- Input – HDL
- Odin
- Logic Optimize
- Technology Map
- Place
- Route
- Output – FPGA bitstream
Heterogeneous FPGA CAD Flow

- Input – HDL
- Odin
- Logic Optimize
- Technology Map
- Place
- Route
- Output – FPGA bitstream
Heterogeneous FPGA CAD Flow

- Input – HDL
- Odin
- Logic Optimize
- Technology Map
- Place
- Route
- Output – FPGA bitstream
Mapping to Heterogeneity

- Partial mapping maps functionality to hard structures on FPGAs

Step 1: Identification Algorithm
Step 2: Binding Algorithm
Partial Mapping Identification Step

Input:
1. Netlist:
2. Library describing hard structure:

[HETEROGENEOUS FPGA LIBRARY diagram]
Identification Goal

• Identify all portions of the circuit that could make use of some version of the hard structure on the FPGA
Identification Output

- Netlist with identified functionality
Algorithm

For each \((f = \text{function available on hard structure})\)

\(seed = \text{a unique part of the } f\)

For each \((e = \text{element in netlist of type seed})\)

RECORD if \((\text{match}(f, e))\)
Partial Mapping Binding

- Input: Netlist with identified functionality
Binding - Goal

• Decide how each element in the netlist will be mapped on the FPGA
Binding - Output

- Netlist with each element bound to soft fabric or hard structure
Binding Algorithm

• Map all identified functionality to FPGA implementations
  – Only find 2 advanced structures in our benchmarks
Mapping Optimizations

• Flip-flops
  – Need to consider the clear signal

• Adder
  – Map to dedicated adder logic on FPGA
RTL Optimizations

- Arithmetic Optimizations
- One-hot Re-encoding of Finite State Machines
- Multiplexer Collapsing
Arithmetic Optimizations

• Goal = Map to smaller arithmetic units
  – Constant propagation
  – Downstream tools don’t always do this for arithmetic structures
Arithmetic Optimizations - Examples

\[
\begin{array}{cccccc}
\text{a[3]} & \text{a[2]} & \text{a[1]} & \text{a[0]} & \text{b[1]} & \text{b[0]} \\
+ \quad b[1] & b[0] & 0 & 0
\end{array}
\]

\[
\begin{array}{cccccc}
\text{o[3]} & \text{o[2]} & \text{o[1]} & \text{o[0]} \\
\end{array}
\]
Arithmetic Optimizations - Examples


One-hot re-encoding of Finite State Machine

• One-hot encoded FSM use less routing and less logic [Golson93]
• Odin identifies and re-encode FSMs
Identifying State Machines in Verilog

always @ (w or CS)
begin
  case (CS)
    A : if (w == 0) NS = A;
        else NS = B;
    B : if (w == 0) NS = B;
        else NS = A;
  endcase
end

always @(posedge clock)
begin
  CS <= NS;
end

Combinational case statement. CS is the state register

All assignments to state Register are constant or feedback loop
Convert to one-hot

- Once identified change the size of the state register and re-encode constants
Problems with FSM re-encoding

- Not guaranteed to find all state machines
Verification

• Odin has been tested by simulating and verifying results through modelsim
  – cf_cordic
  – fir_scu_rtl
  – molecular dynamics
Quality of Results
Basic Goal

• Show Odin produces comparable results to an industrial front-end synthesis tool
Experimental Setup

• Compare Odin against Altera’s Quartus front-end synthesis
  – Use Quartus back-end

• FPGA Used: Stratix I

• CAD Flow based on Altera Quartus 4.1
CAD Flows

- Comparison CAD flows
- Both use Quartus back-end
Benchmarks

• Collection of Verilog Benchmarks
  – Opencores.org
  – SCU-RTL
  – Texas-97
  – Benchmark Suite for Placement-2001
  – Local designs converted from VHDL
    • Raytrace
    • Molecular Dynamics
    • Stereo Vision
## Results – Area Comparison

<table>
<thead>
<tr>
<th>Designs</th>
<th>Ratio</th>
<th>Designs</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>fft_258_6</td>
<td>1.34</td>
<td>reed_sol_decoder1</td>
<td>1.03</td>
</tr>
<tr>
<td>iir1</td>
<td>1.73</td>
<td>reed_sol_decoder2</td>
<td>1.09</td>
</tr>
<tr>
<td>iir</td>
<td>1.14</td>
<td>md</td>
<td>1.41</td>
</tr>
<tr>
<td>fir_3_8_8</td>
<td>1.00</td>
<td>cordin_8_8</td>
<td>1.42</td>
</tr>
<tr>
<td>fir_24_16_16</td>
<td>1.00</td>
<td>cordin_18_18</td>
<td>1.45</td>
</tr>
<tr>
<td>fir_scu_rtl</td>
<td>0.55</td>
<td>MAC1</td>
<td>0.98</td>
</tr>
<tr>
<td>diffeq_f_systemC</td>
<td>1.23</td>
<td>MAC2</td>
<td>0.99</td>
</tr>
<tr>
<td>diffeq_paj_convert</td>
<td>0.72</td>
<td>CRC33_D264</td>
<td>1.00</td>
</tr>
<tr>
<td>sv_chip1</td>
<td>0.97</td>
<td>des_area</td>
<td>0.88</td>
</tr>
<tr>
<td>sv_chip2</td>
<td>1.02</td>
<td>des_perf</td>
<td>0.84</td>
</tr>
<tr>
<td>sv_chip2_no_mem</td>
<td>0.98</td>
<td>sv_chip0</td>
<td>1.02</td>
</tr>
<tr>
<td>rt_raygentop</td>
<td>1.02</td>
<td>sv_chip0_no_mem</td>
<td>0.98</td>
</tr>
<tr>
<td>rt_raygentop_no_mem</td>
<td>1.33</td>
<td>sv_chip3_no_mem</td>
<td>0.79</td>
</tr>
<tr>
<td>rt_top</td>
<td>1.14</td>
<td>rt_frambuf_top</td>
<td>1.44</td>
</tr>
<tr>
<td>rt_top_no_mem</td>
<td>1.37</td>
<td>rt_frambuf_top_no_mem</td>
<td>1.19</td>
</tr>
<tr>
<td>oc45_cpu</td>
<td>1.42</td>
<td>rt_boundtop</td>
<td>1.59</td>
</tr>
</tbody>
</table>

- Ratio < 1 means results from Odin smaller
Results Summary

- Ratio < 1 means results from Odin better
- Geometric Average
  - Area Ratio: 1.11
  - Speed Ratio: 1.05
Effectiveness of RTL Optimizations

- Show how techniques improve results generated by Odin
Results – DSP block reduction

• Arithmetic Optimizations cause all improvement
  – 27.8% less DSP-blocks
## Results – LE Reduction

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Percent Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial Mapping</td>
<td>3%</td>
</tr>
<tr>
<td>Arithmetic optimizations</td>
<td>5%</td>
</tr>
<tr>
<td>One hot state reencoding</td>
<td>11%</td>
</tr>
<tr>
<td>Multiplexer Collapsing</td>
<td>81%</td>
</tr>
</tbody>
</table>

- Overall 4% improvement
### Results – Speed Improvement

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Percent Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial Mapping</td>
<td>13%</td>
</tr>
<tr>
<td>Arithmetic optimizations</td>
<td>15%</td>
</tr>
<tr>
<td>One hot state reencoding</td>
<td>27%</td>
</tr>
<tr>
<td>Multiplexer Collapsing</td>
<td>45%</td>
</tr>
</tbody>
</table>

- Overall 5.6% improvement
Summary

• Odin generates results comparable to Quartus
  – Give numbers
• Showed the relative importance of RTL optimizations
Summary

• Odin available:
  – Open Source
  – GPL software license
Future Work

• Architect better hard structures that are more widely usable